Super-Tasks, Accelerating Turing Machines and Uncomputability

Oron Shagrir
Philosophy Department, The Hebrew University,
Jerusalem 91905, Israel
E-mail address: shagrir@cc.huji.ac.il

Abstract

Accelerating Turing machines are abstract devices that have the same computational structure as Turing machines, but can perform super-tasks. I argue that performing super-tasks alone does not buy more computational power, and that accelerating Turing machines do not solve the halting problem. To show this, I analyze the reasoning that leads to Thomson's paradox, point out that the paradox rests on a conflation of different perspectives of accelerating processes, and conclude that the same conflation underlies the claim that accelerating Turing machines can solve the halting problem.

Keywords: Accelerating Turing machines; The halting problem; Thomson's paradox; Super-tasks.

1. Introduction

A super-task involves carrying out infinitely many actions during a finite interval of time. One sort of super-tasks is implicit in Zeno's paradoxes. Another is suggested by Blake [3], Weyl [27], and Russell [20], who all consider the same form of temporal pattering. Weyl, for example, conceives a machine that would complete "an infinite sequence of distinct acts of decision within a finite time; say by supplying the first result after 1/2 minute, the second after another 1¼ minute, the third 1/8 minute later than the second, etc." (p. 42). Our concern here is with *accelerating Turing machines*. These are abstract devices that have the same static and operational structure as Turing machines, but perform super-tasks by exhibiting the same temporal pattering described by Weyl. They are discussed in Boolos and Jefferey [4], Copeland [7,8,9], Hamkins and Lewis [15], and Stewart [22].

Unlike the 'ordinary' Turing machines, accelerating Turing machines can complete infinitely many steps within a finite span of time. But do they have more computational power? Do they solve, for example, the halting problem? I argue that they do not. There are accelerating devices that compute the halting function, but none is a Turing machine.

2. Accelerating Turing machines

The notion of a Turing machine, introduced by Turing [25] in his classic 1936 paper, is often taken to be *the* model of a computing machine. The details of the machine are well-known, but a statement of Turing is useful for our forthcoming polemic:

> [The machine] is only capable of a finite number of conditions $q_1, q_2, \ldots, q_R$ which will be called "*m*-configurations". The machine is supplied with a "tape"…running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the *r*-th, bearing the symbol $\xi(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol"…. The possible behaviour of the machine at each moment is determined by the *m*-configurations q_n and the scanned symbol $\xi(r)$. This pair $q_n, \xi(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine… If at each stage the motion of a machine… is *completely* determined by the configuration, we shall call the machine an "automatic machine" (or *a*-machine)… In this paper I deal only with automatic machines, and will therefore often omit the prefix *a*-. (pp. 117-118).

According to this characterization, a computational process is a sequence of stages, and the machine's configuration – i.e., the condition of the machine, the position of the scanner and the contents of the tape – at any stage $\alpha + 1$ is completely determined by the machine's configuration at the previous stage α. Similarly, we can see the process as a sequence of state-transition actions (see, e.g., Gandy [13]), whereas 'state' is taken in a broad sense, referring to the machine's configuration. I will henceforth use the terms 'configuration' and 'state' interchangeably.

As Copeland [9, p. 282] observes, there is not much mentioning of time in Turing's 1936 characterization of the machines. There is a listing of the primitive operations, but no specification of their *duration*. The implicit assumption in the literature on computation is that there is a lower bound on duration of a primitive operation. In particular, it is assumed that a process that consists of infinitely many stages – in those cases when the machine never halts – requires infinite time. I will refer to the machines that satisfy this assumption as *ordinary* Turing machines. *Accelerating* Turing machine are ordinary Turing machines save one difference. Their sequence of stages exhibits the temporal pattering mentioned above. Copeland [9] who coined the name accelerating Turing machines describes them as "Turing machines that perform the second primitive operation called for by the program in half the time taken to perform the first, the third in half the time taken to perform the second, and so on" (p. 283). This pattering enables the accelerating machines to complete the same sequence of actions within a finite time, and so to perform super-tasks that ordinary, non-accelerating, machines cannot perform. There are, of course, many devices that can do things that no Turing machine can compute (for a useful survey see Copeland and Sylvan [10]). Even Turing [26] conceives such devices, which he calls machines with oracles, or o-machines. Accelerating Turing machines are of special interest because they preserve the computational structure of a Turing machine, and yet seem to perform tasks that no ordinary Turing machine can do.

One such task is generating the infinite decimal expansion of π: "Since a Turing machine can be programmed to compute π, an accelerating Turing machine can execute each act of writing that is called for by this program before two moments of operating time have elapsed. That is to say, for every n, the accelerating Turing machine writes down the nth digit of the decimal representation of π within two moments of operating time" (Copeland [9, p. 284]). Another task would be to arrive at the truth-values of open questions in mathematics such as Wittgenstein's question of whether or not there exist three consecutive 7s in the decimal representation of π, or the question of whether or not Goldbach's conjecture is true (Pitowsky [19]; Copeland [9]). Consider the latter conjecture, which asserts that every even number larger than two is the sum of two primes. The accelerating machine would operate as follows: its first action is to print on the designated output square the symbol 1 (TRUE). This action requires one minute. It then systematically and successively examines whether an even number is a sum of two primes. Simplifying matters, we could assume it takes 1/2 minute to test out 4, another 1/4 minute to test out 6, 1/8 minute to test out 8, and so forth. If the machine finds a counterexample, i.e., an even number that is not a sum of two primes, it would replace the 1 with 0 (FALSE). If it does not find one, exhausting, as it were, all natural numbers, the machine would never alter the 1 in the output square. One way or another, the machine would complete its task within two minutes, providing the truth-value of Goldbach's conjecture.

Our focus here is on accelerating Turing machines whose task is to compute functions that cannot be computed by any ordinary Turing machine! A classic example is the halting function H(x,y), which characterizes the halting states of Turing machines. H(x,y) returns 1 if the machine whose index is x (in some enumeration of the set of Turing-machines) halts when operating on input y, and returns 0 if the machine never halts. Turing [25] proved that no ordinary Turing machine computes this function; no Turing machine, that is, solves the halting problem. But here is an accelerating device that does. The device is a universal machine that operates as follows. Its first action is to print 0 in the designated output square. Operating as a universal Turing machine, our device then mimics the actions of the xth Turing machine operating on input y. The actions are performed in an accelerated fashion, such that after each operation our device tests whether the mimicked machine arrived at a halting state. If arrived at a halting state, the device replaces the 0 with 1 at the output square and halts. If not, the device keeps working ad infinitum, never replacing the 0 in the designated output square. Either way, the device would tell within two minutes whether the xth Turing machine, operating on input y, halts or not for any given x and y. For a more detailed description of the machine, see Copeland [8, p. 31].

There are arguments, however, against the physical and conceptual possibility of super-tasks in general and accelerating machines in particular (for a critical survey, see Laraudogoitia [18]). On the physical side, Benacerraf and Putnam [2, p. 20] point out that super-tasks are physically impossible because relativity theory sets the velocity of light as a limit on the speed at which the machine's parts can move. This point is echoed in Gandy [13] who argues that there is a bound on the speed of signal propagation in any physical computing device. But it turns out that there are spacetime structures consistent with Einstein’s equations, where superasks are quite easy to come by (Pitowsky [19]; Hogarth [16,17]; but see also Earman and Norton [11]). In these spacetime structures we can devise a non-accelerating o-machine that computes functions that are not Turing machine computable, including the halting function (see also Shagrir and Pitowsky [21]). Yet Benacerraf and Putnam's point does apply to the accelerating machines, whose processes seem to be kinematically impossible. There must be an n, for which it is physically impossible to perform an action in less than $1/2^n$ of a minute. Even in the context of Newtonian mechanics, where the equations enable the body's velocity to increase without bound, this body would disappear from the universe. So in the best case scenario, the accelerating machine would disappear by the end of the process, leaving behind the correct results written on the memory tape alone (see also Copeland [9, p. 289]).

On the conceptual side, Thomson [23] argues that super-tasks are *conceptually impossible* or at least that "the concept of super-task has not been *explained*" (p. 6).

We shall immediately turn to discuss the argument in length, in section 3. Another conceptual puzzle concerns the accelerating device that computes the halting function. On the one hand, this device is, seemingly, "a Turing machine fair and square" (Copeland [9, p. 295]). On the other hand, it computes a function that cannot be computed by any Turing machine. This certainly looks like "a blatant contradiction" (Copeland [9, p. 295]). I return to discuss this puzzle in section 4. My aim here is to show that the two conceptual paradoxes are linked, and that their resolution proceeds along the same lines.

3. Thomson's paradox

Thomson [23] argues that there are "reasons for supporting that super-tasks are not possible of performance" (p. 5). The gist of his argument is that we cannot intelligibly determine what would be the state of the system after the completion of the super-task. Thomson's main example involves a reading lamp, but it can easily be extended to accelerating machines (Thomson himself applies it to π-machines). Consider an accelerating machine that produces, successively, the partial sums of the infinite series 1,-1,1,-1,… At the end of the first stage, which lasts one minute, the machine prints out 1. At the second stage, lasting 1/2 minute, the machine replaces the 1 by 0. At the third stage, lasting 1/4 minute the machine prints 1 instead of the 0, and so forth, ad infinitum. In short, the machine alternates from 0 to 1 and back again unboundedly. What would be printed on the tape after two minutes, when the machine completes its super-task? Is it 1 or 0? It cannot be 1, because the machine always prints 0 immediately after. And it cannot be 0, because the machine always prints 1 immediately after. So, on the one hand, the printout after two moments must be either 1 or 0. But, on the other, the printout can be neither 1 nor 0. What is the way out of this paradoxical situation? The assumption that there is a non-halting machine that computes the partial sums of the diverging series is surely innocent. We must therefore blame the idea of a super-task which seems to be self-contradictory, or at least has to be explained.

Thomson's argument looks neat and convincing. But, as Benacerraf [1] demonstrates, it is invalid. Thompson specified the logical setup of the machine at every stage before two minutes have elapsed. Given this specification, we can tell what would be the actions of the machine during that time period, when the super-task is performed. Yet nothing follows from this specification about the state of the machine at the two moments point and after. It is like the function that is specified to be continuous in the segment [0,2), but is not specified at 2. The function might preserve its continuity at 2, but might lose it as well. All values at 2 would be consistent with that specification. Likewise, the printout on the machine's tape after two minutes can be 0, 1, 17, or nothing at all. Each printout is consistent with the specification of the machine.

It is widely regarded that Benacerraf's critique is successful; even Thomson [24] later admitted that his own argument is worthless. But the critique is hardly relieving. Given how simple Benacerraf's reply is, we might wonder why Thomson's argument looked so neat and convincing in the first place; "why so many naturally conclude otherwise and as a result believe that a contradiction is straining to emerge" (Earman and Norton [12, p. 238]). Furthermore, we are still puzzled about the state of the accelerating machine *after* the super-task is accomplished. As Copeland [9] puts it, "Thomson's query as to what state an infinity machine may consistently be supposed to be in *after* it completes its super-task is a good one" (p. 286). It is a good query because we are still convinced that the printout on the machine's tape after two minutes depends on the prior history of the machine, even if we haven't specified what the machine would do after two minutes. So we wonder what could be printed out on the tape given that the previous printouts form the diverging infinite sequence 1,0,1,0,….

I suggest that we are puzzled because we are still drawn in by a picture that conflates different perspectives of the accelerating machine. One is the Turing machine (TM) perspective. According to this perspective, the accelerating machine is just a Turing machine that computes, by executing a suitable program, the partial sums of the diverging infinite series, and thus alternately prints 1 and 0 and back unboundedly. From that point of view, this acceleration process is an infinite sequence of stages, whereas the machine's configuration (state) at each stage is completely determined by the configuration of previous stage. According to a different perspective, which I call the *physical* perspective, we view the machine in terms of its physical makeup. From that perspective, the machine produces physical tokens of 1s and 0s because its dynamics, governed by the laws of physics, dictate this behavior. From this point of view, too, each state of the machine depends on the prior history of the machine, only that here the relevant history is the *physical* one. In particular, if the machine (or just the tape) somehow survives the acceleration process, its state at the two-minute stage is dictated by its prior physical states.[1]

Thomson's paradox emerges when we conflate the two perspectives. Taking the physical perspective, we assert that the machine is in some state after two minutes. Taking the TM perspective, we assert that any machine's state – its configuration at this stage – is completely determined by the previous state. We thus conclude that the machine's configuration after two minutes is dictated by *the* previous Turing machine's configuration. Yet we cannot intelligibly tell what configuration precedes the two-minute stage. For, as Thomson convincingly shows, if it printed 1, the machine prints 0 immediately after, and if it printed 0, the machine prints 1 immediately after. We thus cannot tell what would be the printout on the tape after two minutes either. Thomson concludes that the fault is to be found in the assumption such a machine is conceptually possible. And since the idea of a non-halting machine surely makes sense, Thomson concludes that the "talk of super-tasks is *senseless*" [23, p. 9].

But the argument is flawed. The flaw is in taking the machine's state, at the second minute, to be not only a physical state, *but also a Turing machine's state*, whereas, the Turing machine at that time is no longer specified. The device, as a Turing machine, completes all its actions *before* the second moment. Even if the device survives the acceleration, its physical state after two minutes is no longer a Turing machine's state. All the infinitely many states of the pertinent Turing machine were implemented at the time period prior to the second minute. But if the machine's state, after two minutes, is not a Turing machine's state, it need not be dictated any longer by *the* previous state of the (Turing) machine we specified. From the TM perspective, *any* machine's state after two minutes would be *perfectly consistent* with the accelerating Turing machine that was in action during the time segment [0,2). Inconsistency emerges nowhere.

Setting the physical and TM perspectives apart, we see that no paradox emerges. But we still wonder about the machine's *physical* state after two moments, after the super-task had been accomplished. We wonder whether we can consistently retain all the following assertions about the physical setting of the machine: (1) that the machine is in some physical state after completing the super-task; (2) that each state is dictated by the prior physical history; and (3) that this physical history consists of alternating printing 1's and 0's on the tape. But as Earman and Norton [12, pp. 237-

239] point out, inconsistency emerges only when we implicitly introduce another assumption, about persistence. The assumption is that the information on the machine's tape is left unchanged until the next printing (physical) operation. In particular, the printout on the designated output square would be 0 as long as it was not replaced by 1, and vice versa. This persistence property amounts to requiring that the information in the designated output square after two moments is the limit of printouts in that square prior to the second moment [12, p. 238]. Assuming *that*, there cannot be an answer as to what would be printed on that square after two minutes, simply because there is no limit to the series consisting of the states of the tape, prior to that time. Any attempt to construct an accelerating device that successively produces 1's and 0's and that uses the persistence property must fail. "The machine must be constructed to satisfy an inconsistent specification. This is clearly impossible in any consistent physical setting." [12, p. 239]. However, we certainly cannot conclude that all other specifications of super-task machines are also inconsistent. We can certainly specify, as we did in section 1, *other* accelerating machines that, arguably, satisfy persistence. And we can specify an accelerating machine that alternately prints 0's and 1's, unboundedly, yet does not satisfy persistence at the two-minute stage. What would be the printout on the tape after two minutes, after the super-task is accomplished? It is certainly an interesting empirical question. But given that persistence is not satisfied, it is no longer a puzzling one.

4. Computing the uncomputable

Let us now turn to the other paradox: that of accelerating Turing machines that compute functions that no Turing machine can compute. How are we to solve *this* paradox? One way out of the conundrum is to deny that such a machine computes at all. Another is to distinguish between different senses of computing. Thus Copeland [9] argues that the halting problem refers to computation in the internal sense, and that the accelerating Turing machine computes the halting function only in the external sense. I do not contest these solutions but offer another instead. I urge that the same reasoning that led to Thomson's paradox – a conflation between TM and physical perspectives – also creates the current paradox, and so that the same reasoning that liberated us from Thomson's paradox also applies in the current case. In particular, if we accept Benacerraf's critique of Thomson's paradox, as we should, we must give up the idea that the machines that compute the halting function, generating the infinite expansion of π, etc., are Turing machines.

Consider again the accelerating device that computes the halting function. Recall how it works: it first prints 0, then simulates the acts of the nth machine receiving m as an input, and replacing the 0 by 1 just in case the simulated machine halts at some point. When the super-task is completed, after two minutes, we have at our disposal the halting status of the simulated machine. But who is computing the (Turing) incomputable halting function? If we take Benacerraf's critique seriously, then the answer is that it is certainly not a Turing machine. We did specify an accelerating Turing machine that simulates the acts of the nth Turing machine (for any n and m), yet all the acts of this Turing machine take place *before* the second minute. Nothing follows from this specification about the state of the device *after* two minutes, when the super-task is completed. Any printout on the machine's tape after two minutes, be it 0, 1, or 17, is perfectly consistent with the halting program that the machine executed before that. If the accelerating Turing machine does not halt, there is no point during the acceleration, at the time segment [0,2), at which the super-task has been accomplished. And after two minutes, when the super-task has been finally completed, the state of the (Turing) machine is no longer specified.

So why do we take the device to compute the halting function? The reason, I maintain, is that we also look at the device from another, most commonly a *physical*, perspective. From that perspective, we assume, rightly or wrongly, that the device exists after two minutes, and we also assume, rightly or wrongly, that the device's tape persists in its current physical state as long as there is no printing operation. Taken together, we assume that the printout on the tape after two minutes is the limit of the prior printouts on the tape, at the time of acceleration. And given that these printouts represent the halting state of the nth machine acting on input m, we take this limit to be no less than the (physical) representation of the solution of the halting problem.

A paradox emerges only when we conflate the TM and the physical perspectives. Taking the TM perspective, we conceive the device as an accelerating Turing machine that executes the program described above. Taking the physical perspective, we maintain that the device is in some state after two minutes. We thus conclude that the device's state after two moments is a Turing machine's state that represents the value of a function that no Turing machine can compute. We are relieved when we realize that the device's state after two minutes is, at best, a physical state, and not a Turing machine's state. We thus should conclude that if anything computes the halting function, it is the *physical* device, not the Turing machine. The accelerating Turing machine computes exactly the same function computed by a non-accelerating Turing machine. It returns 1 if the simulated nth machine halts, and never returns a value if the nth machine never halts.

The same reasoning applies to the other accelerating machines. Assuming that Goldbach's conjecture is true, the (accelerating) Turing machine will systematically check out every even number. Yet nothing about the specification of this Turing machine dictates what would be the state of the device after two minutes. Any printout on the tape, be it 1 (TRUE) or 0 (FALSE) or nothing at all, is consistent with the specified Turing machine. No Turing machine generates the infinite decimal expansion of π either. The accelerating Turing machine prints on the tape any digit of the infinite decimal expansion. But at no point during the acceleration the task is completed, and when the task had been accomplished, after two minutes, the device is no longer the specified Turing machine.

Of course that we can *extend* the Turing machine concept so that it will encompass the second-minute stage. We can specify the value of the designated output at that moment to be the limit of the previous values that this cell has displayed. Such specification is offered by Hamkins and Lewis [14,15], who introduce the concept of *an infinite time Turing machine*. This machine preserves the static structure of an ordinary machine. Its successive steps of computation also proceed in the classical manner: "the classical procedure determines the configuration of the machine… at any stage $\alpha + 1$, given the configuration at any stage α" [14, p. 256]. What is new is the behavior of the machine at the transfinite domain. At any limit ordinal stage, "the machine is placed in the special *limit* state, just another of the finitely many states; and the values in the cells of the tapes are updated by computing a kind of limit of the previous values that cell has displayed" [14, p. 526]. We can thus have an accelerating infinite time Turing machine that computes the halting function. The machine operates like a universal machine before two minutes have elapsed. After printing 0 at the output square, it simulates the operations of the nth machine operating on input m, replacing the 0 with 1 just in case the simulated machine halts. During this period the machine behaves in the classical manner, meaning that the configuration of each stage is completely determined by the configuration of the previous stage. So far there is not much difference from the accelerating Turing machine described above. But there is a difference. The infinite time machine also encompasses the two-minute stage. This stage is a ω_1 limit stage, in which the value at the designated output cell is the limit of the previous values in that cell, namely, displaying the halting state of the nth machine, acting on input m.[2]

I do not deny that this infinite time Turing machine computes the halting function. I also do not mind the name infinite time *Turing machine*. Rather, my point is this. If accelerating Turing machines have exactly the same computational structure as ordinary Turing machines, then they compute exactly the Turing machine computable functions. Performing super-tasks enables the accelerating machines to complete infinitely many steps in a finite interval of time, but it does not enable to compute functions that the ordinary machines cannot compute. And if accelerating Turing machines differ from ordinary Turing machines in computational structure, as is the case with infinite time Turing machines, then they might have more computational power. But here too, the difference in computational power is not due to performing super-tasks alone. Performing a super-task only ensures that the computation terminates in a finite *real* time, even if it requires infinitely many computation steps. The difference in computational power owes to the difference in computational end structure. Either way, no paradox emerges. If the accelerating Turing machine has the same computational structure as the ordinary machine, it does not compute the halting function. And if we extend the concept of the Turing machine, redefining the end structure, it should come as no surprise that the newly specified Turing machines compute functions, e.g., the halting function, that Turing's machines – the machines that Turing specified – fail to compute.

Acknowledgements: I am thankful to Mark Burgin, Jack Copeland, Yuval Dolev, Itamar Pitowsky, Carl Posy and Michael Roubach for discussion and comments. This research was supported by The Israel Science Foundation (grant no. 857/03-07).

References

[1] P. Benacerraf, Tasks, super-tasks, and the modern elastics, Journal of Philosophy 59 (1962) 765-784.
[2] P. Benacerraf, H. Putnam, eds., Philosophy of Mathematics, Selected Readings, Cambridge University Press, Cambridge, 1964.
[3] R.M. Blake, The paradox of temporal process, , Journal of Philosophy 23 (1926) 645-654.

[4] G.S. Boolos, R.C. Jeffrey, Computability and Logic, third ed., Cambridge University Press, Cambridge, 1989.
[5] M.S. Burgin, Universal limit Turing machines, Notices of the Russian Academy of Sciences, 325 No. 4 (1992) 654-658.
[6] M.S. Burgin, How we know what technology can do, Communications of the ACM, 44 No. 11 (2001) 82-88.
[7] B.J. Copeland, Even Turing machines can compute uncomputable functions, in C. Claude, J. Casti, M. Dinneen, eds., Unconventional Models of Computation, Springer-Verlag, London, 1998, 150-164.
[8] B.J. Copeland, Super Turing-machines, Complexity 4 (1998) 30-32.
[9] B.J. Copeland, Accelerating Turing machines, Minds and Machines 12 (2002) 281-301.
[10] B.J Copeland, R. Sylvan, Beyond the universal Turing machine, Australasian Journal of Philosophy 77 (1999) 46-67.
[11] J. Earman, J.D. Norton, Forever is a day: Supertasks in Pitowsky and Malament-Hogarth spacetimes, Philosophy of Science 60 (1993) 22-42.
[12] J. Earman, J.D. Norton, Infinite pains: The trouble with supertasks, in A. Morton, S.P. Stich, eds., Benacerraf and his Critics, Blackwell, Oxford, 1996, 231-261.
[13] R.O. Gandy, Church's thesis and principles of mechanisms, in J. Barwise, J.J. Keisler, K. Kunen, eds., The Kleene Symposium, North-Holland, Amsterdam, 1980, 123-145.
[14] J.D. Hamkins, Infinite time Turing machines, Minds and Machines 12 (2002) 521-539.
[15] J.D. Hamkins, A. Lewis, Infinite time Turing machines, Journal of Symbolic Logic 65 (2000) 567-604.
[16] M.L. Hogarth, Does general relativity allow an observer to view an eternity in a finite time?, Foundations of Physics Letters 5 (1992) 173-181.
[17] M.L. Hogarth, Non-Turing computers and non-Turing computability, Proceedings of the Philosophy of Science Association 1 (1994) 126-138.
[18] J.P. Laraudogoitia, Supertasks, in J. Perry, E. Zalta, eds., Stanford Encyclopedia of Philosophy, [http://plato.stanford.edu], 1999.
[19] I. Pitowsky, The physical Church thesis and physical computational complexity, Iyyun 39 (1990) 81-99.
[20] B.A.W. Russell, The limits of empiricism, Proceedings of the Aristotelian Society 36 (1936) 131-150.
[21] O. Shagrir, I. Pitowsky, Physical hypercomputation and the Church–Turing thesis, Minds and Machines 13 (2003) 87-101.
[22] I. Stewart, Deciding the undecidable, Nature 352 (1991) 664-665.
[23] J.F. Thomson, Tasks and super-tasks, Analysis 15 (1954) 1-13.
[24] J.F. Thomson, Comments on Professor Benacerraf's paper, in W.C. Salmon, ed., Zeno's Paradoxes, Bobbs-Merrill, Indianapolis, 1970, 130-138.
[25] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proceedings of the London Mathematical Society (2) 42 (1936) 230-265. A correction in 43 (1937) 544-546. Reprinted in M. Davis, ed., The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, Raven, New York, 1965, 115-154. Page numbers refer to the 1965 edition.
[26] A.M. Turing, Systems of logic based on ordinals, Proceedings of the London Mathematical Society (2) 45 (1939) 161-228. Reprinted in M. Davis, ed., The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, Raven, New York, 1965, 154-222.
[27] H. Weyl, Philosophie der Mathematik und Naturwissenschaft, R. Oldenbourg, Munich, 1927. Page numbers refer to the English translation: Philosophy of Mathematics and Natural Science, Princeton University Press, Princeton, 1949.

Notes

[1] I would like to emphasize that we could easily replace the physical perspective by any other which asserts that the machine is in some state *after* accomplishing the super-task. I refer to the physical perspective both because many (see, e.g., Earman and Norton [12]; Copeland [9]) discuss the accelerating processes from that perspective, and because it is easy to conflate the physical and TM perspectives. It is easy because, along the acceleration process, the physical device *implements* the Turing machine. Similarly, we could replace the TM perspective by any other which implies that the machine is a rule-following device, namely, that each machine's state is determined by the previous one. In Thomson's main example – with the reading lamp – the 'TM perspective' implies that each OFF-state is followed by an ON-state, and vice versa. The 'physical perspective' implies that the lamp is in some state after two minutes.

[2] Other constructions, which retain the static structure of Turing machines, modifying only the end structure, can be made in terms of *inductive Turing machines* (Burgin [6]) and *limit Turing machines* (Burgin [5]). These machines, too, can compute the halting function.